# Untitled

August 27, 2019

[ ]:
```
# library setup
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
from scipy import stats
import random
import statsmodels.api as sm
from scipy.stats import pearsonr
from sklearn import preprocessing
from statsmodels.formula.api import ols
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from math import sqrt
from sklearn import metrics
from numpy import cov
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
pd.set_option('display.float_format', lambda x:'%f'%x)
```

[ ]:
```
os.chdir("/home/santosh")
```

[ ]:
```
os.getcwd()
```

[ ]:
```
excel_file='Project.xls'
```

[ ]:
```
absent = pd.read_excel(excel_file)
```

[ ]:
```
absent['Absenteeism time in hours'].value_counts()
```

[ ]:
```
# Missing Value Analysis
missing_val = pd.DataFrame(absent.isnull().sum())
```

[ ]:
```
missing_val = missing_val.reset_index()
```

```
[ ]: missing_val = missing_val.rename(columns = {'index':'variables',0:
     ↪'Missing_Percentage'})

[ ]: missing_val['Missing_Percentage'] = (missing_val['Missing_Percentage']/
     ↪len(absent))*100

[ ]: missing_val

[ ]: # Imputing missing values with help of mean and median
     absent['Reason for absence'] = absent['Reason for absence'].
     ↪fillna(absent['Reason for absence'].median())

[ ]: absent['Month of absence'] = absent['Month of absence'].fillna(absent['Month of␣
     ↪absence'].median())

[ ]: absent['Transportation expense'] = absent['Transportation expense'].
     ↪fillna(absent['Transportation expense'].median())

[ ]: absent['Distance from Residence to Work'] = absent['Distance from Residence to␣
     ↪Work'].fillna(absent['Distance from Residence to Work'].median())

[ ]: absent['Service time'] = absent['Service time'].fillna(absent['Service time'].
     ↪median())

[ ]: absent['Age'] = absent['Age'].fillna(absent['Age'].median())

[ ]: absent['Work load Average/day '] = absent['Work load Average/day '].
     ↪fillna(absent['Work load Average/day '].median())

[ ]: absent['Hit target'] = absent['Hit target'].fillna(absent['Hit target'].
     ↪median())

[ ]: absent['Disciplinary failure'] = absent['Disciplinary failure'].
     ↪fillna(absent['Disciplinary failure'].median())

[ ]: absent['Education'] = absent['Education'].fillna(absent['Education'].median())

[ ]: absent['Social drinker'] = absent['Social drinker'].fillna(absent['Social␣
     ↪drinker'].median())

[ ]: absent['Social smoker'] = absent['Social smoker'].fillna(absent['Social␣
     ↪smoker'].median())

[ ]: absent['Son'] = absent['Son'].fillna(absent['Son'].median())

[ ]: absent['Pet'] = absent['Pet'].fillna(absent['Pet'].median())
```

[ ]:
```
absent['Height'] = absent['Height'].fillna(absent['Height'].median())
```

[ ]:
```
absent['Weight'] = absent['Weight'].fillna(absent['Weight'].median())
```

[ ]:
```
absent['Body mass index'] = absent['Body mass index'].fillna(absent['Body mass␣
↪index'].mean())
```

[ ]:
```
absent['Absenteeism time in hours'] = absent['Absenteeism time in hours'].
↪fillna(absent['Absenteeism time in hours'].median())
```

[ ]:
```
absent.isnull().sum()
```

[ ]:
```
data = absent.copy()
```

[ ]:
```
absent['ID'] = absent['ID'].astype('category')
absent['Reason for absence'] = absent['Reason for absence'].astype('category')
absent['Month of absence'] = absent['Month of absence'].astype('category')
absent['Day of the week'] = absent['Day of the week'].astype('category')
absent['Seasons'] = absent['Seasons'].astype('category')
absent['Disciplinary failure'] = absent['Disciplinary failure'].
↪astype('category')
absent['Education'] = absent['Education'].astype('category')
absent['Social drinker'] = absent['Social drinker'].astype('category')
absent['Social smoker'] = absent['Social smoker'].astype('category')
```

[ ]:
```
#numeric = absent[['Transportation expense', 'Distance from Residence to Work',␣
↪'Service time', 'Age', 'Work load Average/day ', 'Hit target',
    # 'Son', 'Pet', 'Weight', 'Height', 'Body mass index', 'Absenteeism time in␣
↪hours']]
```

[ ]:
```
#numeric.shape
#factor = absent[['ID', 'Reason for absence', 'Month of absence', 'Day of the␣
↪week','Seasons', 'Disciplinary failure', 'Education', 'Social drinker',
#       'Social smoker']]
```

[ ]:
```
# outlier analysis
get_ipython().run_line_magic('matplotlib', 'inline')
plt.boxplot(absent['Transportation expense'])
```

[ ]:
```
plt.boxplot(absent['Distance from Residence to Work'])
```

[ ]:
```
plt.boxplot(absent['Service time'])
```

[ ]:
```
plt.boxplot(absent['Age'])
```

[ ]:
```
plt.boxplot(absent['Work load Average/day '])
```

[ ]:
```
plt.boxplot(absent['Hit target'])
```

[ ]:
```
plt.boxplot(absent['Son'])
```

[ ]:
```
plt.boxplot(absent['Pet'])
```

[ ]:
```
plt.boxplot(absent['Weight'])
```

[ ]:
```
plt.boxplot(absent['Height'])
```

[ ]:
```
plt.boxplot(absent['Body mass index'])
```

[ ]:
```
plt.boxplot(absent['Absenteeism time in hours'])
```

[ ]:
```
for i in absent :
    print(i)
    q75,q25 = np.percentile(absent.loc[:,i],[75,25])
    iqr = q75 - q25
    min = q25 - (iqr*1.5)
    max = q75 + (iqr*1.5)

    print(min)
    print(max)
```

[ ]:
```
# calculating minimum and maximum values
q75,q25 = np.percentile(absent['Transportation expense'],[75,25])
```

[ ]:
```
iqr = q75 - q25
```

[ ]:
```
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)
print(minimum)
print(maximum)
```

[ ]:
```
absent.loc[absent['Transportation expense']< minimum,:'Transportation expense']␣
 ↪= np.nan
absent.loc[absent['Transportation expense']> maximum,:'Transportation expense']␣
 ↪= np.nan
```

[ ]:
```
q75,q25 = np.percentile(absent['Age'],[75,25])
```

[ ]:
```
iqr = q75 - q25
```

[ ]:
```
minimum2 = q25 - (iqr*1.5)
maximum2 = q75 + (iqr*1.5)
print(minimum2)
print(maximum2)
```

[ ]:
```
absent.loc[absent['Age']< minimum2,:'Age'] = np.nan
absent.loc[absent['Age']> maximum2,:'Age'] = np.nan
```

[ ]:
```
q75,q25 = np.percentile(absent['Service time'],[75,25])
```

[ ]:
```
iqr = q75 - q25
```

[ ]:
```
minimum = q25 - (iqr*1.5)
maximum = q75 + (iqr*1.5)
print(minimum)
print(maximum)
```

[ ]:
```
absent.loc[absent['Service time']< minimum,:'Service time'] = np.nan
absent.loc[absent['Service time']> maximum,:'Service time'] = np.nan
```

[ ]:
```
q75,q25 = np.percentile(absent['Work load Average/day '],[75,25])
```

[ ]:
```
iqr = q75 - q25
```

[ ]:
```
minimum3 = q25 - (iqr*1.5)
maximum3 = q75 + (iqr*1.5)
print(minimum3)
print(maximum3)
```

[ ]:
```
absent.loc[absent['Work load Average/day ']< minimum3,:'Work load Average/day␣
↪'] = np.nan
absent.loc[absent['Work load Average/day ']> maximum3,:'Work load Average/day␣
↪'] = np.nan
```

[ ]:
```
q75,q25 = np.percentile(absent['Hit target'],[75,25])
```

[ ]:
```
iqr = q75 - q25
```

[ ]:
```
minimum4 = q25 - (iqr*1.5)
maximum4 = q75 + (iqr*1.5)
print(minimum4)
print(maximum4)
```

[ ]:
```
absent.loc[absent['Hit target']< minimum4,:'Hit target'] = np.nan
absent.loc[absent['Hit target']> maximum4,:'Hit target'] = np.nan
```

[ ]:
```
q75,q25 = np.percentile(absent['Pet'],[75,25])
```

[ ]:
```
iqr = q75 - q25
```

[ ]:
```
minimum6 = q25 - (iqr*1.5)
maximum6 = q75 + (iqr*1.5)
print(minimum6)
print(maximum6)
```

[ ]:
```
absent.loc[absent['Pet']< minimum6,:'Pet'] = np.nan
absent.loc[absent['Pet']> maximum6,:'Pet'] = np.nan
```

[ ]:
```
q75,q25 = np.percentile(absent['Height'],[75,25])
```

[ ]:
```
iqr = q75 - q25
```

[ ]:
```
minimum8 = q25 - (iqr*1.5)
maximum8 = q75 + (iqr*1.5)
print(minimum8)
print(maximum8)
```

[ ]:
```
absent.loc[absent['Height']< minimum8,:'Height'] = np.nan
absent.loc[absent['Height']> maximum8,:'Height'] = np.nan
```

[ ]:
```
# imputing outliers values with median
absent['Transportation expense'] = absent['Transportation expense'].
 ↪fillna(absent['Transportation expense'].median())
absent['Age'] = absent['Age'].fillna(absent['Age'].median())
absent['Work load Average/day '] = absent['Work load Average/day '].
 ↪fillna(absent['Work load Average/day '].median())
absent['Hit target'] = absent['Hit target'].fillna(absent['Hit target'].
 ↪median())
absent['Service time'] = absent['Service time'].fillna(absent['Service time'].
 ↪median())
absent['Pet'] = absent['Pet'].fillna(absent['Pet'].median())
absent['Height'] = absent['Height'].fillna(absent['Height'].median())
absent['Absenteeism time in hours'] = absent['Absenteeism time in hours'].
 ↪fillna(absent['Absenteeism time in hours'].median())
```

[ ]:
```
# Copying data in new object "data"
absent['ID'] = data['ID']
absent['Reason for absence'] = data['Reason for absence']
absent['Month of absence'] = data['Month of absence']
absent['Day of the week'] = data['Day of the week']
absent['Seasons'] = data['Seasons']
absent['Distance from Residence to Work'] = data['Distance from Residence to␣
 ↪Work']
absent['Disciplinary failure'] = data['Disciplinary failure']
absent['Education'] = data['Education']
absent['Son'] = data['Son']
```

```
absent['Social drinker'] = data['Social drinker']
absent['Social smoker'] = data['Social smoker']
absent['Weight'] = data['Weight']
absent['Body mass index'] = data ['Body mass index']
```

[ ]:
```
# checking missing values after outlier analysis
missval = pd.DataFrame(absent.isnull().sum())
```

[ ]:
```
missval
```

[ ]:
```
# Converting data in proper data types
absent['ID'] = absent['ID'].astype('category')
absent['Reason for absence'] = absent['Reason for absence'].astype('category')
absent['Month of absence'] = absent['Month of absence'].astype('category')
absent['Day of the week'] = absent['Day of the week'].astype('category')
absent['Seasons'] = absent['Seasons'].astype('category')
absent['Disciplinary failure'] = absent['Disciplinary failure'].
 ↪astype('category')
absent['Education'] = absent['Education'].astype('category')
absent['Social drinker'] = absent['Social drinker'].astype('category')
absent['Social smoker'] = absent['Social smoker'].astype('category')
```

[ ]:
```
# feature selection
numeric_c = absent[['Transportation expense', 'Distance from Residence to␣
 ↪Work', 'Service time', 'Age', 'Work load Average/day ', 'Hit target',
     'Son', 'Pet', 'Weight', 'Height', 'Body mass index','Absenteeism time in␣
 ↪hours']]
```

[ ]:
```
# Feature selection
corr = numeric_c.corr()
```

[ ]:
```
f,ax = plt.subplots(figsize = (10,8))
sns.heatmap(corr,mask = np.zeros_like(corr,dtype = np.object),cmap = sns.
 ↪diverging_palette(220,10,as_cmap = True),square = True, ax=ax,annot = True)
```

[ ]:
```
# anova for categorical variable
factor = absent[['ID', 'Reason for absence', 'Month of absence', 'Day of the␣
 ↪week','Seasons', 'Disciplinary failure', 'Education', 'Social drinker',
       'Social smoker',]]
```

[ ]:
```
print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Reason for␣
 ↪absence"]))
```

[ ]:
```
print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Month of␣
 ↪absence"]))
```

```
[ ]: print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Day of the␣
     ↪week"]))

[ ]: print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Seasons"]))

[ ]: print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Disciplinary␣
     ↪failure"]))

[ ]: print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Education"]))

[ ]: print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Social␣
     ↪drinker"]))

[ ]: print(stats.f_oneway(absent["Absenteeism time in hours"],absent["Social␣
     ↪smoker"]))

[ ]: data = absent.copy()

[ ]: absent = absent.drop(['ID','Seasons','Education','Height','Hit␣
     ↪target','Pet','Body mass index','Disciplinary failure','Age','Social␣
     ↪smoker','Social drinker','Son'],axis = 1)

[ ]: absent.shape

[ ]: # DAta normalisation
     #Normality check
     absent['Transportation expense'].hist(bins = 20)

[ ]: absent['Distance from Residence to Work'].hist(bins = 20)

[ ]: absent['Service time'].hist(bins = 20)

[ ]: absent[ 'Work load Average/day '].hist(bins = 20)

[ ]: absent['Weight'].hist(bins = 20)

[ ]: # Data Normalisation
     from sklearn.preprocessing import normalize
     normalized_absent = preprocessing.normalize(absent)

[ ]: absent.dtypes

[ ]: # ML Algorithm
     ## dividing data into train and test
     train,test = train_test_split(absent,test_size= 0.2)
```

[ ]:
```
# Decision Tree Regression
random.seed(123)
fit = DecisionTreeRegressor(max_depth = 2).fit(train.iloc[:,0:8],train.iloc[:
↪,8])
```

[ ]:
```
predictions_dt = fit.predict(test.iloc[:,0:8])
```

[ ]:
```
mse_dt = (mean_squared_error(test.iloc[:,8], predictions_dt))
print(mse_dt)
```

[ ]:
```
rmse_dt = sqrt(mean_squared_error(test.iloc[:,8],predictions_dt))
print(rmse_dt)
```

[ ]:
```
# Random forest
# n = 100
random.seed(123)
rfregressor100 = RandomForestRegressor(n_estimators = 100, random_state = 0)
rfregressor100.fit(train.iloc[:,0:8],train.iloc[:,8])
```

[ ]:
```
predictions_rf100 = rfregressor100.predict(test.iloc[:,0:8])
```

[ ]:
```
mse_rf100 = (mean_squared_error(test.iloc[:,8], predictions_rf100))
print(mse_rf100)
```

[ ]:
```
rmse_rf100 = sqrt(mean_squared_error(test.iloc[:,8],predictions_rf100))
print(rmse_rf100)
```

[ ]:
```
# Random forest for n = 200
random.seed(123)
rfregressor200 = RandomForestRegressor(n_estimators = 200, random_state = 0)
rfregressor200.fit(train.iloc[:,0:8],train.iloc[:,8])
```

[ ]:
```
predictions_rf200 = rfregressor200.predict(test.iloc[:,0:8])
```

[ ]:
```
mse_rf200 = (mean_squared_error(test.iloc[:,8], predictions_rf200))
print(mse_rf200)
```

[ ]:
```
rmse_rf200 = sqrt(mean_squared_error(test.iloc[:,8],predictions_rf200))
print(rmse_rf200)
```

[ ]:
```
# Random forest for n = 300

rfregressor300 = RandomForestRegressor(n_estimators = 300, random_state = 0)
rfregressor300.fit(train.iloc[:,0:8],train.iloc[:,8])
```

[ ]:
```
predictions_rf300 = rfregressor300.predict(test.iloc[:,0:8])
```

[ ]:
```
mse_rf300 = (mean_squared_error(test.iloc[:,8], predictions_rf300))
print(mse_rf300)
```

[ ]:
```
rmse_rf300 = sqrt(mean_squared_error(test.iloc[:,8],predictions_rf300))
print(rmse_rf300)
```

[ ]:
```
# Random forest for n = 500

rfregressor500 = RandomForestRegressor(n_estimators = 500, random_state = 0)
rfregressor500.fit(train.iloc[:,0:8],train.iloc[:,8])
```

[ ]:
```
predictions_rf500 = rfregressor500.predict(test.iloc[:,0:8])
```

[ ]:
```
mse_rf500 = (mean_squared_error(test.iloc[:,8], predictions_rf500))
print(mse_rf500)
```

[ ]:
```
rmse_rf500 = sqrt(mean_squared_error(test.iloc[:,8],predictions_rf500))
print(rmse_rf500)
```

[ ]:
```
# Linear regression

absent['Reason for absence'] = absent['Reason for absence'].astype('float')
absent['Day of the week'] = absent['Day of the week'].astype('float')
absent['Month of absence'] = absent['Month of absence'].astype('float')
```

[ ]:
```
train1,test1 = train_test_split(absent,test_size = 0.2)
```

[ ]:
```
line_regression = sm.OLS(train1.iloc[:,8],train1.iloc[:,0:8]).fit()
```

[ ]:
```
line_regression.summary()
```

[ ]:
```
predictions_lr = line_regression.predict(test1.iloc[:,0:8])
```

[ ]:
```
mse_lr = (mean_squared_error(test.iloc[:,8], predictions_lr))
print(mse_lr)
```

[ ]:
```
rmse_linear = sqrt(mean_squared_error(test1.iloc[:,8],predictions_lr))
print(rmse_linear)
```

[ ]:
```
## LOSS per month
data.shape
```

[ ]:
```
loss = data[['Month of absence','Service time','Work load Average/day␣
 ↪','Absenteeism time in hours']]
```

[ ]:
```
# Work loss = (Workload*Absenteeism time)/Service time

loss["loss_month"] = (loss['Work load Average/day ']*loss['Absenteeism time in␣
↪hours'])/loss['Service time']
```

[ ]:
```
loss.shape
loss.head(5)
```

[ ]:
```
loss["loss_month"] = np.round(loss["loss_month"]).astype('int64')
```

[ ]:
```
No_absent = loss[loss['Month of absence'] == 0]['loss_month'].sum()
January = loss[loss['Month of absence'] == 1]['loss_month'].sum()
February = loss[loss['Month of absence'] == 2]['loss_month'].sum()
March = loss[loss['Month of absence'] == 3]['loss_month'].sum()
April = loss[loss['Month of absence'] == 4]['loss_month'].sum()
May = loss[loss['Month of absence'] == 5]['loss_month'].sum()
June = loss[loss['Month of absence'] == 6]['loss_month'].sum()
July = loss[loss['Month of absence'] == 7]['loss_month'].sum()
August = loss[loss['Month of absence'] == 8]['loss_month'].sum()
September = loss[loss['Month of absence'] == 9]['loss_month'].sum()
October = loss[loss['Month of absence'] == 10]['loss_month'].sum()
November = loss[loss['Month of absence'] == 11]['loss_month'].sum()
December = loss[loss['Month of absence'] == 12]['loss_month'].sum()
```

[ ]:
```
loss.head(5)
```

[ ]:
```
data1 = {'No Absent': No_absent, 'Janaury': January,'Febraury':␣
↪February,'March': March,
      'April': April, 'May': May,'June': June,'July': July,
      'August': August,'September': September,'October': October,'November':␣
↪November,
      'December': December}
```

[ ]:
```
workloss = pd.DataFrame.from_dict(data1,orient = 'index')
```

[ ]:
```
workloss.rename(index = str, columns={0:"Workload loss pr month"})
```